

SDEV1201 Database Fundamentals

LESSON 11

Let's review

Entity Relationship Model

Entity

An entity is a person, place, thing or concept that is used in the business context of the application and for which data is collected.

Attributes

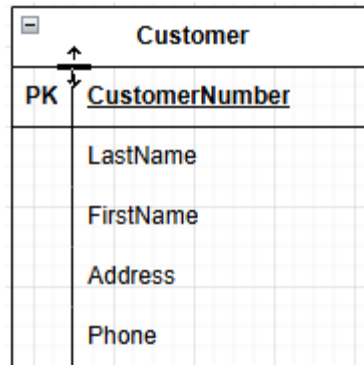
A piece of data that describes an entity is called an **attribute**. Synonyms for attribute include: element, property and field. In the order example, the customer entity has the following attributes:

- Customer Number
- Last Name
- First Name
- Address
- Phone Number

The values for all attributes describing an entity make up one instance of the entity. For example, the following values describe one specific customer.

Customer Number	Last Name	First Name	Address	Phone
100	Adams	Joan	9825-77 Ave Edm Ab T5C 8E2	488-8712

Entities and Tables



After the ERD design is complete, when the database is created the entities become tables.

<u>CustomerNumber</u>	FirstName	<u>LastName</u>	Address	Phone
100	Adams	Julie	9825-77 Ave Edmonton AB T5C 8E2	780-111-1111
200	Smith	Miles	4308-133 Ave Edmonton AB T5C 8E9	780-222-2222

What Are Data Types?

Unlike humans, a computer does not know the difference between "1234" and "abcd." A data type is a classification that dictates what a variable or object can hold in computer programming. Data types are an important factor in virtually all computer programming languages. Each data type requires different amounts of memory and has specific operations that can be performed over it.

CREATE TABLE **minimum syntax**

11

```
CREATE TABLE TableName (  
    Column1 DATATYPE  
    , Column2 DATATYPE  
    , Column3 DATATYPE  
    ...  
)
```

where *Column#* represents the name of the attribute/column and *DATATYPE* is *INT*, *DATETIME*, etc.

Activity

Work on question 1 on the “Create Table Exercises PostgreSQL” exercise found in “Week 5 – Create Table” in Brightspace in the Activities section. The “Funky Flowers ERD” (located in the same location) will be needed for this exercise.

Create Table Exercise Solution

No Keys

-- Create Tables

Create Table Vehicles

```
(  
    VehicleNumber integer not null,  
    DriverFirstName varchar (35) not null,  
    DriverLastName varchar (35) not null  
);
```

Create Table Deliveries

```
(  
    DeliveryNumber integer not null,  
    DeliveryDate date not null,  
    VehicleNumber integer not null  
);
```

Create Table Exercise Solution

No Keys

Create Table Customers

```
(  
    CustomerNumber int not null,  
    FirstName varchar (35) not null,  
    LastName varchar (35) not null,  
    StreetAddress varchar (35) not null,  
    City varchar (35) not null,  
    Province char (2) not null,  
    PostalCode char (6) not null,  
    PhoneNumber char (12) not null  
);
```

Create Table Products

```
(  
    ProductNumber char (7) not null,  
    Description varchar (50) not null,  
    CurrentPrice decimal (7,2) not null  
);
```

Create Table Exercise Solution

No Keys

Create Table Orders

```
(  
  OrderNumber integer not null,  
  OrderDate date not null,  
  FloristNumber integer not null,  
  CustomerNumber integer not null,  
  GSTAmount decimal (7,2) not null,  
  DeliveryNumber integer not null,  
  DeliveredYN char (1) not null,  
  PaidYN char (1) not null  
);
```

Create Table OrderDetails

```
(  
  OrderNumber integer not null,  
  ProductNumber char (7) not null,  
  Quantity integer not null,  
  SellingPrice decimal (7,2) not null  
);
```

Today's Objective

By the end of this section you will be able to understand:

- ❑ Primary and Foreign Key Constraints.
- ❑ Indexes.

PostgreSQL Cast Operator

The PostgreSQL CAST function provides an efficient way to convert data types in PostgreSQL, which is important when ensuring data is in the correct format for storage, calculations, or comparisons.

In PostgreSQL, we can use CAST to transform data between various data types, such as converting strings to integers, dates, or booleans. This article will guide you through the PostgreSQL CAST syntax, examples of its usage, and important considerations.

`CAST ( expression AS target_type );`

Key Terms

- **Expression:** The value to be converted. This can be a constant, a table column, or any expression that evaluates to a value.
- **Target_Type:** The data type to which the expression is to be converted.

PostgreSQL Cast Operator Examples

```
SELECT CAST ('100' AS INTEGER);
```

```
SELECT  
    CAST ('2020-01-01' AS DATE),  
    CAST ('01-OCT-2020' AS DATE);
```

```
Select Cast(Supplier_ID As varchar), Company_Name  
From Suppliers
```

Constraints

Constraints are used to:

21

1. Define the **PK**
 - This is defined by the **primary key** constraint
2. Define **relationships**
 - This is defined by the **foreign key** constraint
3. Define **default values**
 - Defined by the **default** constraint
4. Define a **domain of valid values**
 - Defined by the **check** constraint
5. Ensure that **all values in a column are unique**

Constraints

Constraints can be defined:

1. When the table is initially created, as part of the Create Table statement or,
2. For an existing table, via the Alter Table statement.

We use the Constraint keyword to define a constraint. A single column can have multiple constraints, each with its own constraint keyword definition. Each constraint must have a unique name. The constraint name is used in SQL messages, so use descriptive constraint names. We will use a prefix, as part of the constraint name, to identify the type of constraint:

Prefix Constraint Type

PK Primary Key

FK Foreign Key

CK Check

DF Default

UQ Unique

Primary Key Constraint

Primary Key Constraint

When using a normalized database design, all tables must have a primary key constraint. Any column that acts as a primary key must be defined as a not null column. The DBMS will ensure that all rows in the table have a unique value for primary key columns.

When a primary key constraint is defined, PostgreSQL automatically creates a unique index for the column(s) acting as the primary key.

Syntax

Column constraint syntax:

[Constraint constraint_name] Primary Key

Table constraint syntax:

[Constraint constraint_name]

Primary Key (col_name [, col_name2 [, . . . col_name32]])

Primary Key syntax: column-level constraint

Syntax:

[Constraint constraint_name] Primary Key

Create Table Student

(

StudentID char (9) not null Constraint PK_Student Primary Key,

LastName varchar (20) not null,

FirstName varchar (15) not null

);

PostgreSQL Identity Property

Syntax:

GENERATED BY DEFAULT AS IDENTITY

Create Table Student

(

StudentID integer not null Constraint PK_Student Primary Key GENERATED BY DEFAULT AS IDENTITY,

LastName varchar (20) not null,

FirstName varchar (15) not null

);

Primary Key syntax: table-level constraint

What about tables with composite keys?

Example:

Create Table Marks

(

StudentID char (9) not null,

CourseID char (6) not null,

Mark integer null,

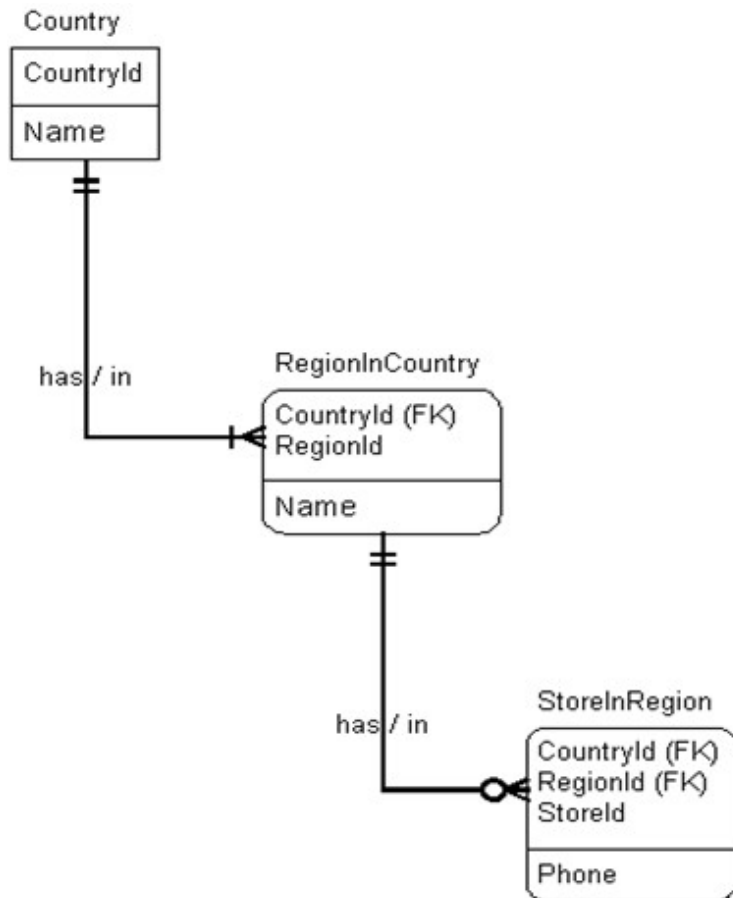
Constraint PK_Marks Primary Key (StudentID, CourseID)

);

Foreign Key constraint

Foreign Key Constraint

29



The FK constraint defines a **relationship** between rows, and defines a **parent-child relationship** between tables.

This affects:

1. Creating tables
2. Dropping tables
3. Inserting/updating/deleting rows in tables

A child record/table cannot exist without its parent.

FK Constraints & referential integrity

The foreign key constraint defines referential integrity between two tables. Operations that affect the tables and/or the data stored within the tables must comply with referential integrity. This affects:

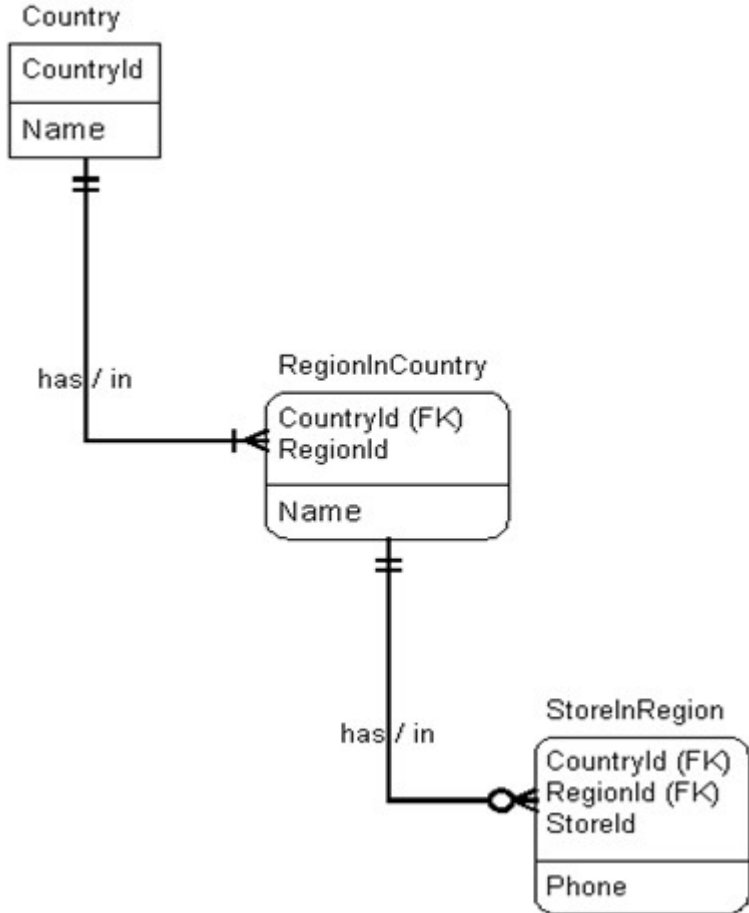
1. Dropping tables.
2. Creating tables.
3. Inserting/Updating/Deleting rows in tables.

The bottom line for referential integrity is that a child row/table cannot exist without its associated parent row/table. Therefore:

1. You must Drop a child table BEFORE you Drop the associated parent table.
2. You must Create a parent table BEFORE you Create the associated child table.
3. The value of a column acting as a foreign key must be:
 - A value that exists as a primary key in the associated parent table, or
 - NULL

For example

31



Drop

1. DROP TABLE StoreInRegion
2. DROP TABLE RegionInCountry
3. DROP TABLE Country

Create

4. CREATE TABLE Country (...)
5. CREATE TABLE RegionInCountry (...)
6. CREATE TABLE StoreInRegion (...)

Foreign Key Constraint Syntax

Column constraint syntax:

[Constraint constraint_name]
References ref_table (ref_column)

Table constraint syntax:

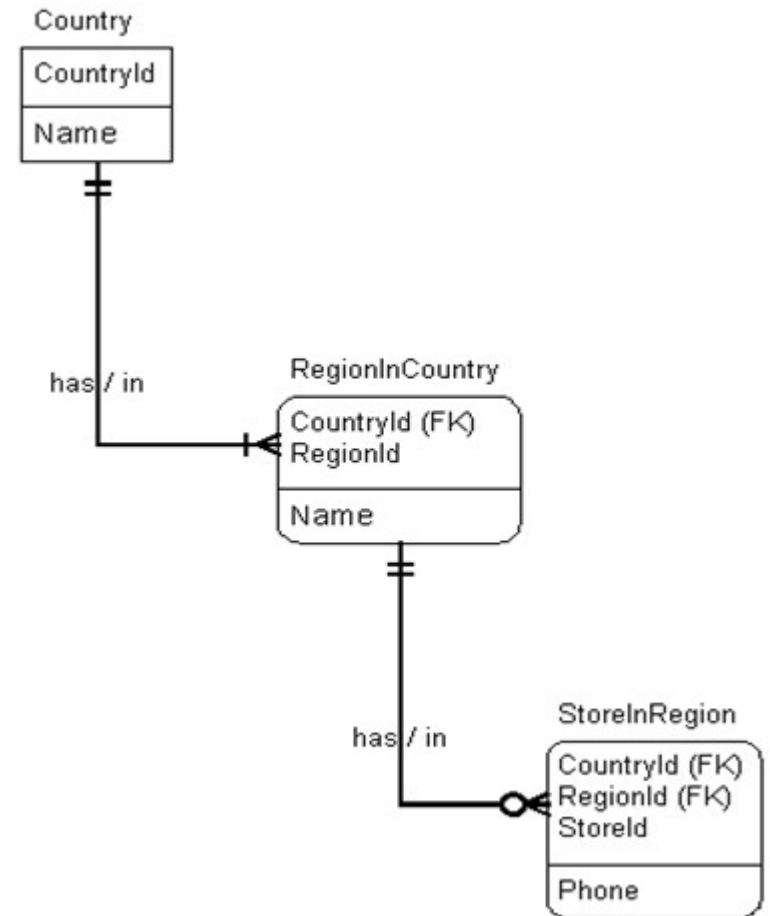
[Constraint constraint_name]
Foreign Key (column_name [, . . . column_name32])
References ref_table (ref_column [, . . . ref_column32])

- column_name identifies the column(s) acting as the foreign key
- ref_table identifies the parent table
- ref_column identifies the primary key in the associated parent table

RegionInCountryExample

33

```
Create Table RegionInCountry
(
    CountryID integer not null
    Constraint FK_RegionInCountry_Country
    References Country (CountryID),
    RegionID integer not null,
    Name varchar (100) not null,
    ConstraintPK_RegionInCountry_CountryID_RegionID
    Primary Key (CountryID, RegionID)
);
```

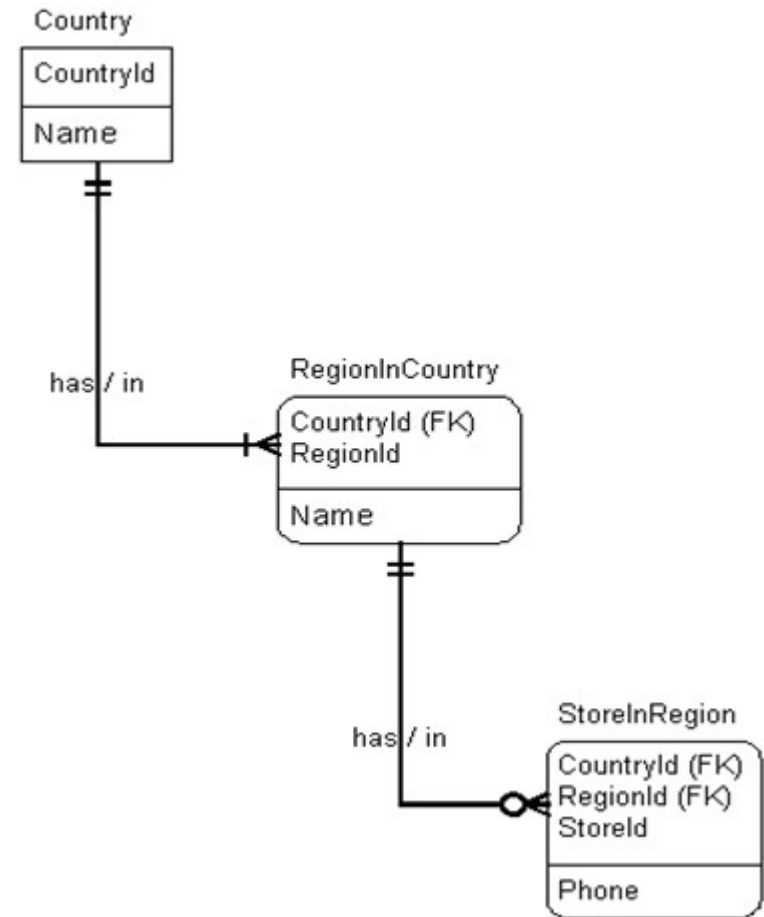


StoreInRegion Example

34

Create Table StoreInRegion

```
(  
  CountryID integer not null,  
  RegionID integer not null,  
  StoreID integer not null,  
  Phone varchar (100) not null,  
  Constraint PK_CountryId_RegionId_StoreID  
    Primary key (CountryID, RegionID, StoreID),  
  Constraint FK_StoreInRegion_RegionInCountry  
    Foreign Key (CountryID, RegionID)  
      References RegionInCountry (CountryID, RegionID)  
);
```



Indexes

Indexes

An index is an object stored in the db.

It's associated with 1 or more columns, in a specific table, that act as **the key for the index**.

This helps the DBMS look up (retrieve) the data quicker.

Functionally an index object is similar to an index at the back of a book.

Primary Key Indexes

PostgreSQL automatically creates an index for each unique constraint and primary key constraint to enforce uniqueness. Thus, it is not necessary to create an index explicitly for primary key columns. Each entry in the index contains information (pointers) to where the associated row in the table is located. This information is then used by the DBMS to speed up retrieval of data from the table.

Indexes

IndexID "points" to the RowID in the actual table

Table				
RowID	PersonID	FirstName	LastName	More columns...
1	510	Ken	Sanchez	
2	515	Terri	Duffy	
3	565	Roberto	Tamburello	
4	535	Rob	Walters	
5	500	Gail	Erickson	
6	560	Jossef	Goldberg	
7	530	Dylan	Miller	
8	545	Diane	Goldberg	
9	525	Gigi	Matthew	
10	505	Michael	Sullivan	
11	550	Ovidiu	Cracium	
12	570	Thierry	D'Hers	
13	540	Janice	Galvin	
14	520	Michael	Rahe	
15	555	Sharon	Salavaria	

Index (PersonID)	
IndexID	PersonID
5	500
10	505
1	510
2	515
14	520
9	525
7	530
4	535
13	540
8	545
11	550
15	555
6	560
3	565
12	570

Index (Last Name, First Name)		
IndexID	LastName	FirstName
11	Cracium	Ovidiu
12	D'Hers	Thierry
2	Duffy	Terri
5	Erickson	Gail
13	Galvin	Janice
8	Goldberg	Diane
6	Goldberg	Jossef
9	Matthew	Gigi
7	Miller	Dylan
14	Rahe	Michael
15	Salavaria	Sharon
1	Sanchez	Ken
10	Sullivan	Michael
3	Tamburello	Roberto
4	Walters	Rob

Index (First Name, Last Name)		
IndexID	FirstName	LastName
8	Diane	Goldberg
7	Dylan	Miller
5	Gail	Erickson
9	Gigi	Matthew
13	Janice	Galvin
6	Jossef	Goldberg
1	Ken	Sanchez
14	Michael	Rahe
10	Michael	Sullivan
11	Ovidiu	Cracium
4	Rob	Walters
3	Roberto	Tamburello
15	Sharon	Salavaria
2	Terri	Duffy
12	Thierry	D'Hers

Non-Primary Key Index

A non-primary key index provides a method of using a second criterion for quickly finding/retrieving rows from a table. For example, we could define a non-unique index for the Registration table that uses StaffID as the key of the index. The DBMS could use this index to quickly retrieve the rows for the staff involved in a specific registration.

You can, theoretically, have an unlimited number of indexes per table only constrained by practical constraints like storage and performance. There is a maximum of 32 columns per index.

Considerations

- Indexes **speed up data retrieval**, but **slow down operations** to modify data in a table.
- Consider a table with a clustered index and 3 non-clustered indices.
 - To insert a new row, the DBMS must add the row AND update four indices.
 - To update a row: the DBMS deletes the old row, updates each index, adds a new row, and updates each index again. Updating the index might mean adding/deleting several items.
 - That's a lot of overhead!

Choosing a key

- Some designers create a non-clustered index for each FK. This increased overhead needs to be weighed against the improved performance of retrieving data.

Syntax

```
Create [Unique] Index [If Not Exists] index_name  
    On table_name ( column_name [, ...n] [ Asc | Desc] );
```

Column name represents the key for the index. One or more columns may act as the key for the index. All indexes must have a unique name. In SDEV1201 we normally use the prefix **IX** when naming an index.

Example

Example: create a non-unique index associated with the **Registration** table using **StaffID** (a foreign key referencing the Staff table) as the index key:

```
Create Index IX_Registration_StaffID  
On Registration (StaffID);
```

Example

Example: create a unique index associated with the **Club** table using **ClubName** (to ensure no club names are re-used) as the index key:

```
Create Unique Index IX_Club_ClubName  
    On Club (ClubName);
```

Drop Index Syntax

The **Drop** Index statement deleted an index object. Note: you cannot **drop** an index used for a Primary Key.

Drop Index [If Exists] index_name;

Example: drop the Registration StaffID index just created

Drop Index IX_Registration_StaffID;

Practice

Work on question 2 on “01 Create Table PostgreSQL” (found in “Week 6 – Alter Table & Constraints” in Brightspace)

Reminder

Take-home Coding Assignment Advanced SELECT is due on Friday, October 10, 2025 at 5:00 PM.